

17. Съставни типове данни

Проф. д-р Емил Хаджиколев

1. Структура
2. Клас
3. Обединение
4. Изброим тип
5. Колекции

Структура

- Структурата е съставен тип данни.
- Структурата е **организация на разнотипни елементи**.
- При създаване на конкретни обекти от типа на структурата, елементите получават конкретни стойности.
- **Елементите на структурата се наричат полета и се декларират като променливи.**
- Чрез структури се описват **физическите характеристики на сложни обекти, явления, процеси и др. от реалния свят.**
- Примери за обекти и физическите им характеристики (в даден компютърен модел):
 - Точка в двумерно (декартово) пространство: координати x,y;
 - Човек: име, презиме, фамилия, ЕГН;
 - Студент: име, презиме, фамилия, ЕГН, факултетен номер, оценки;
 - Продукт: име, цена, описание;
- Някои от физическите характеристики (очевидно) може да са съставни обекти.

Деклариране на структура в C++

```
struct <име на типа за структура>{  
    <тип 1> <променливи 1>;  
    <тип 2> <променливи 2>;  
    ...  
    <тип n> <променливи n>  
}[списък с променливи от типа];
```

- След типовете на полетата се описва една променлива или няколко, разделени със запетаи.
- За полетата може да се задават стойности по подразбиране.
- След тялото на структурата задължително се задава символа „точка и запетая“ (;
- Променливи от типа за структура се дефинират като обикновени променливи: **Person p**;
- Достъп до елементите на променлива от структура се извършва с оператор „точка“ (.): **p.name**.
- Може след тялото на структурата и преди „точка и запетая“ да опишем списък с променливи за нея.

Пример за работа със структура в C++ (1)

```
#include <iostream>
#include <string>

using namespace std;

// Дефиниране на структура
struct Point2D {
    double x;
    double y;
};
```

Пример за работа със структура в C++ (2)

```
// Функция, която връща като низ информация за точка от тип Point2D, зададена като
// параметър.

string pointAsString(Point2D point) {
    string asString = "(" + to_string(point.x) + ", " + to_string(point.y) + ")";
    return asString;
}

// Функция, която връща разстоянието между две точки от тип Point2D, зададени като
// параметър.

// Прилага се Питагоровата теорема.

double distance(Point2D p1, Point2D p2) {
    double dist = sqrt(pow(p1.x-p2.x, 2) + pow(p1.y - p2.y, 2));
    return dist;
}
```

Пример за работа със структура в C++ (3)

```
int main() {
    setlocale(LC_ALL, "bg");

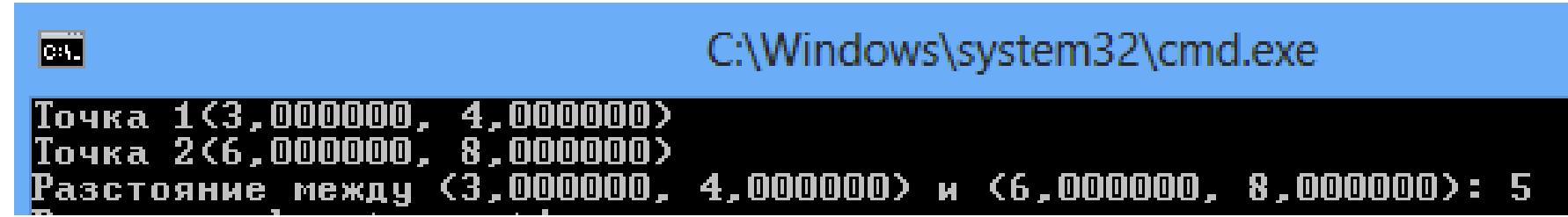
    Point2D p1; // Променливи за точка от тип Point2D
    Point2D p2;

    p1.x = 3; // Задаване на стойности на полетата
    p1.y = 4;

    p2.x = 6;
    p2.y = 8;

    // Извеждане на информация
    cout << "Точка 1<(3,0000000, 4,0000000)>\n";
    cout << "Точка 2<(6,0000000, 8,0000000)>\n";
    cout << "Разстояние между <(3,0000000, 4,0000000)> и <(6,0000000, 8,0000000)>: " <<
        distance(p1, p2) << '\n';

    return 0;
}
```



Инициализация на обект за структура

- При инициализацията на обект за структура се задават стойности на полетата.
- Не може да използваме не-инициализирани полета. Например, грешка е:

```
Point2D p2;  
cout << "p2.x: " << p2.x << '\n'; // грешка
```

- Инициализация на обект:
 - задаване на стойности на полетата в кода

```
p2.x = 1; p2.y = 2*3;
```
 - автоматично задаване на „нулеви стойности“ чрез т.нар. конструктор по подразбиране (подробности за конструкторите по ООП)

```
Point2D p1 = Point2D(); // p1.x и p1.y ще имат стойности 0
```

При това, в последствие трябва отново да бъдат зададени коректни стойности за полетата.

Инициализация на обект за структура. Пример

```
// създаване на статичен обект за Point2D
// чрез конструктор и асоциирането му с променливата p1;
// полетата се инициализират с 0;
Point2D p1 = Point2D();

// само се заделя памет за p2, но не се инициализират полета
Point2D p2;
p2.x = 1; p2.y = 2; // инициализиране на полетата

cout << "p1.x: " << p1.x << '\n'; // 0
cout << "p2.x: " << p2.x << '\n'; // 1
```

Памет и структури

- Броят на байтовете заети от структурите, е сума от байтовете на отделните полета;
- Понякога броят на байтовете се допълва до число кратно на 8 (или друго – 16, 32 – което зависи от компилатора).
- Броят на байтовете за тип или конкретна променлива може да се вземе с функцията (унарния оператор)
`sizeof(<тип/променлива>);`
- Използване като унарен оператор (без скобите):
`sizeof <тип/променлива>;`

Пример за заета памет (1) – Дефиниране на допълнителни структури

```
struct CharAndDoubleStruct {  
    double x;  
    char c;  
};
```

```
struct CharStruct {  
    char c;  
};
```

Пример за заета памет (2) – Проверка за заета памет със sizeof

```
cout << "sizeof(double): " << sizeof(double) << '\n';

// Point2D и p1 от предните слайдове
cout << "sizeof(Point2D): " << sizeof(Point2D) << '\n';
cout << "sizeof(p1): " << sizeof(p1) << '\n';

cout << "sizeof(char): " << sizeof(char) << '\n';

cout << "sizeof(CharStruct): " << sizeof(CharStruct) << '\n';
cout << "sizeof(CharAndDoubleStruct): " << sizeof(CharAndDoubleStruct) << '\n';
```

```
sizeof(double): 8
sizeof(Point2D): 16
sizeof(p1): 16
sizeof(char): 1
sizeof(CharStruct): 1
sizeof(CharAndDoubleStruct): 16
```

Вложени структури. Пример (1)

- Полетата на структурите може да бъдат от всякакви примитивни и съставни типове данни (масиви и други);
- Може да бъде и указател към текущия тип (но не и статична променлива от текущия тип; детайли за указателите в следващата лекция).

```
struct Person {  
    string name;          // име на човека  
    Person *boss;         // указател към шеф - от текущия тип – в C++ не може "Person boss"  
    Point2D location;    // местоположение в някаква двумерна координатна система;  
};  
// Втори вариант  
struct Person {  
    ...  
    struct Point2D {  
        double x, y;  
    } location; // location е поле от тук декларираната структура Point2D  
};
```

Вложени структури. Пример (2)

```
Point2D p1;  
p1.x = 3; p1.y = 4;
```

```
Person b; // За описание на шеф - boss  
b.name = "Иван";  
b.location = p1; // p1 е дефинирана по-рано  
b.boss = NULL; // шефа няма шеф - но може да му се зададе
```

```
Person worker; // За описание на работник  
worker.name = "Петър";  
worker.location.x = 6; // може да не дефинираме обект за точка  
worker.location.y = 8;  
worker.boss = &b; // на указателя worker.boss се присвоява адреса на променливата boss
```

Класове

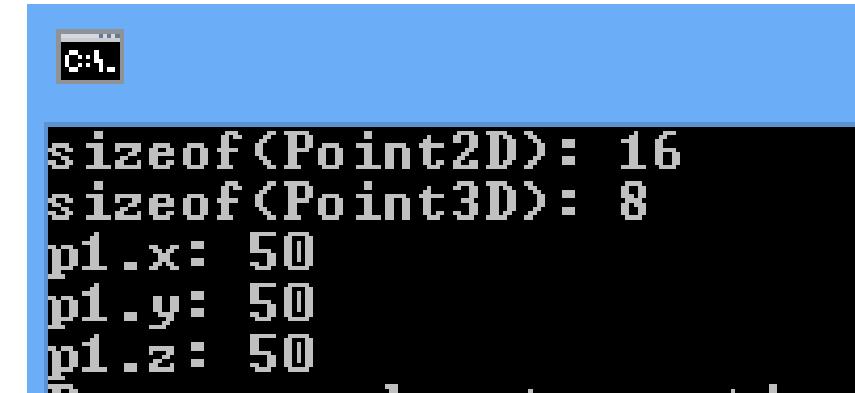
- **Класовете описват физически и функционални характеристики на обекти (свойства, явления, процеси) от реалния свят.**
- Подобни са на структурите, които обаче имат само полета (по дефиниция).
- Функционалните характеристики са методи/функции описани в тялото на клас, които виждат и могат да използват полетата на класа без да се задават като параметри.
- Всеки обект от даден клас има собствени данни и методите работят със съответните конкретни данни.
- В C++ структурите са почти като класовете – в телата им може да се описват методи и да се използват всички възможности на обектно-ориентираното програмиране – наследяване, полиморфизъм, капсулиране, абстракция. Основна разликата е, че по подразбиране полетата и методите в структурите са видими чрез обект (p1.x), докато при класовете – не са видими. Друга разлика е, че при структурите не може да се работи с темплейти.
- Подробности: в дисциплината ООП.

Обединения

- **Обединението е съставен тип данни.**
- Подобно на структурата е **организация на разнотипни елементи, които обаче споделят обща памет.**
- Използват се за съхранение на данни от различни типове по различно време.
- Тъй като паметта за отделните полета е общая, заделената памет (броя на байтовете) е равна на паметта заемана от полето с най-голяма дължина.
- Не може да се описват функции в обединенията.

Пример за обединение

```
union Point3D {  
    double x, y, z;  
};  
  
int main() {  
    cout << "sizeof(Point2D): " << sizeof(Point2D) << '\n';  
    cout << "sizeof(Point3D): " << sizeof(Point3D) << '\n';  
  
    Point3D p1;  
    p1.x = 50; // паметта за полетата x, y, z е една и съща  
    cout << "p1.x: " << p1.x << '\n';  
    cout << "p1.y: " << p1.y << '\n';  
    cout << "p1.z: " << p1.z << '\n';  
    return 0;  
}
```



```
0:~%
```

```
sizeof(Point2D): 16  
sizeof(Point3D): 8  
p1.x: 50  
p1.y: 50  
p1.z: 50
```

Изброим тип - Enumeration

- Изброимият тип описва множество от именувани константи.
- Всяка константа получава целочислена стойност
 - автоматично, като се започва от 0.
 - или се задава от програмиста.
- Променливите от изброим тип могат да получат като стойност само константи от същия тип;
- В програмата достъпът до константите става:
 - чрез името - <константа>
 - квалифицирано име - <тип>::<константа>

Пример за изброим тип

```
enum Colors { RED, GREEN, BLUE };

// нов по-сигурен, но различен от другите ЕП начин - enum class Colors

enum Months {
    JANUARY = 1, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER,
    NOVEMBER, DECEMBER
};

int main() {
    cout << Colors::RED << '\n'; // отпечатване: 0 – квалифицирано име

    Months month = FEBRUARY;      // не-квалифицирано име
    cout << month << '\n';       // отпечатване: 2

    return 0;
}
```

Колекции

- Колекциите обединяват еднотипни данни.
- В колекциите може да има или да няма подредба на елементите.
- В обикновените масиви, в C++, например, които са вид колекция, има наредба на елементите. Наредбата се определя от стойността на ключа/индекса на элемента.
- Може да реализираме самостоятелно клас за колекция – като клас/структура или няколко класа/структури.
- В ЕП има множество стандартни класове, реализиращи различните видове колекции (които няма да разглеждаме).

Основни видове колекции

- Списъци – линейни структури от данни;
- Дървовидни и мрежови структури;
- Масиви (подобни на списъците) – обикновени и асоциативни
- Множества.

Операции върху колекции

- Основните действия върху колекции:
 - добавяне на елемент;
 - извличане на елемент;
 - обхождане на елементите;
 - сортиране.
- Различните видове колекции, притежават различни алгоритми за реализацията на основните операции.

СПИСЪЦИ

- **Списък (list, vector)** – линейна структура от данни, при която елементите са наредени последователно. Може да се добавят и извличат елементи на/от произволно място.
- **Опашка (Queue)** – списък, при който елементите се добавят само в края, а се извличат от началото. Принцип: първи влязъл-първи излязъл (FIFO – First In-First Out);
- **Стек (Stack)** – списък, при който елементите се добавят и извличат от края. Принцип: последен влязъл-първи излязъл (LIFO – Last In-First Out);
- **Дек (Dequeue, Double-Ended Queue, опашка с два края)** – списък, при който елементите могат да се добавят и извличат от началото и от края.
- За реализацията на списъци може да се използват масиви или структури/класове, в които има указатели към предходен и/или следващ елемент.

Дърводидни и мрежови структури от данни

- **Дървета (Tree)** – описват йерархични взаимоотношения между елементите. Примери: структура на директориите, йерархия на служителите във фирма и др.
- **Графи (Net)** – описват множество връзки между елементите.

Кратки (минимални) сведения:

- Може да се добавят елементи (наричани възли) на произволно място в структурата. Първият елемент се нарича корен, крайните – листа;
- Връзките (наричани ребра) също може да са сложни елементи, които носят информация.
- Извличането на елемент става чрез търсене. Има различни техники за търсене целящи по-бързото намиране (не-намиране) на елемент.

Масиви

- Масивът съдържа наредена последователност от еднотипни елементи от вида ключ-стойност;
- Масивите приличат на списъци, но разликата е, че броят на елементите на масива е фиксиран, а на списъка – не.
- Достъпът до стойност на масива става чрез ключ.
- Може да има еднакви стойности за елементите, но ключовете са различни.

Видове масиви

- **Обикновени** – ключовете са последователни цели числа
- **Асоциативни** (hashtable, hashmap, dictionary) – ключовете може да са произволни обекти, обикновено еднотипни. За всеки обект-ключ може да се изчисли т.нар. хеш код – число, което определя обекта по уникален начин.
- Както при обикновените масиви, **при асоциативните масиви ключовете (или съответните им хеш кодове) са уникални.**
- Реализацията на обикновените масиви в ЕП е по-лесна: Индексите може да не се съхраняват реално, защото всички еднотипни елементи заемат еднакъв брой байтове x в паметта. Тогава, i -тият елемент се намира в клетка с адрес $i * x$ след началния адрес на масива.
- Реализацията на асоциативни масиви може да е чрез дърводидни или линейни структури от данни.

Множество - Set

- Множеството е съвкупност от еднотипни елементи, в което няма повторения.